METHOD AND APPARATUS FOR ELIMINATING THE SOFTWARE GENERATED READY-SIGNAL TO HARDWARE DEVICES THAT ARE NOT PART OF THE MEMORY COHERENCY DOMAIN

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The preferred embodiments of the present invention are generally related to increasing computer performance in the realm of software to hardware communications. More particularly, the preferred embodiments of the present invention are directed to a communication technique between software and hardware devices that informs downstream hardware devices that information is available in the system main memory with bounded latency.

Background of the Invention

[0004] In terms of availability and access to data in main memory, computer systems can be broken up into portions within the coherency domain, and portions outside the coherency domain. Some exemplary computer system components that would typically be within the coherency domain are microprocessor caches and the main memory. In a single microprocessor system, the difficulty of maintaining coherency of the microprocessor cache against main memory is relatively simple. Simplicity fades quickly however as multiple microprocessors are included in the

computer system. Generally speaking, the cache of each microprocessor and the main memory array are kept coherent by the use of some form of cache coherency protocol, and the devices are thus within the same coherency domain.

[0005] As frequently happens in a computer system, software executed on one of the microprocessors needs to communicate command lists, data, or both to a downstream hardware device, most likely an input/output device such as a network interface card. In the related art devices, the exchange of command lists and/or data generally takes place by software executed within a microprocessor writing the commands and/or data to a first-in/first-out (FIFO) buffer in main memory. Once all or a significant portion of the commands and/or data have been placed in the FIFO buffer, the software sends a ready signal, also known as a doorbell, to the downstream hardware device indicating that the commands and/or data are available. Once the notification or doorbell has been received, the hardware device arbitrates for mastership of its associated bus, and reads the data from the buffer in main memory through known direct memory access techniques. In this way, the software is free to perform other steps, or the microprocessor may preempt that thread and execute other software threads, while the hardware device reads the commands and/or data, and executes the necessary steps. However, trends in software programming techniques inject the possibility of significant latency between placing of the commands and/or data in the FIFO buffer, and the doorbell notification arriving at the hardware device.

[0006] The standard paradigm in software to hardware communications over the last several years comprises one or more layers of abstraction between the software and the actual hardware device. That is, rather than the software having the capability of writing or communicating directly with the hardware device, software communicates with hardware through a driver program. Thus, it is the driver program or software that is responsible for protocol translations, handshaking, and

the like, to move the commands and/or data from the software thread to the hardware. The advantage of this layer of abstraction from the programming point of view is that the software need only communicate with the appropriate driver, and hardware specific protocols and procedures are not a concern of the higher level program. Of course, the driver software, again just another program executed on the microprocessor, is still responsible for the communication process, including writing to the FIFO buffer and ringing the hardware doorbell as described above.

[0007] Recently, however, the trend has been to write software programs in "user-mode." In user-mode, communications between the software and hardware may take place without levels of abstraction, or may take place with one or more levels of abstraction using drivers in a non-prioritized (non-kernel) mode. Regardless of whether the user-mode software communicates directly with the hardware device, or through a level of abstraction being a driver for that hardware device, software makes the communication. Although any piece of software is susceptible to preemption in today's computer systems, non-kernel software is especially vulnerable to such preemption. By preemption it is meant that for reasons beyond control of the software stream, execution is stopped for a time so that other processes and procedures may take place. These interruptions may be attributable to interrupts directed to the microprocessor, but may also be preemption to execute software streams with higher priority. Regardless of the reason, preemption at the wrong time, with regard to the software-to-hardware communication, has the potential for creating unbounded latencies between placing commands and/or data, and notifying the hardware.

[0008] Consider a related art communication from software executed on a microprocessor to a hardware device by way of a FIFO buffer in main memory. Further consider that the software has the opportunity to write the commands and/or data into the FIFO, but before the software can ring the hardware doorbell (send the message across one or more bridge devices and expansion buses),

the software is preempted for an extended period of time. In this situation, the commands and/or data are loaded, but the hardware has yet to act because it has not received notification.

[0009] Preemption between the loading of the FIFO and the ringing of the hardware doorbell is possible whether the program is a user-mode program, an abstracted level of user-mode communication, or even a kernel mode driver. Inopportune preemption, however, is more prevalent in the user-mode and abstracted user-mode communications.

[0010] Thus, what is needed in the art is a more efficient way to notify hardware that commands and/or data are available in the buffer that facilitates communication between the software running on a microprocessor in the coherent memory domain and the hardware.

BRIEF SUMMARY OF SOME OF THE PREFERRED EMBODIMENTS

[0011] The problems noted above are solved in large part by a system and related method whereby hardware devices are allowed to participate in the coherency domain, preferably on a limited basis. More particularly, the hardware devices are preferably equipped with a cache memory that duplicates a small subset of the main memory, that subset being the location of the FIFO buffer. This small cache type memory on the hardware device is preferably maintained coherent with the locations in main memory through the cache coherency protocol of the computer system. In this way, software programs need only load commands and/or data in the FIFO buffer in main memory (or more particularly write those locations in the caches of the microprocessors in which they execute), and the cache coherency protocol notifies the hardware device by invalidating the shared memory locations stored in the cache memory of the hardware device. The notification that commands and/or data are available for the hardware device is thus accomplished by means of the invalidate command of the cache coherency protocol.

[0012] Implementing notification in this manner, the hardware device need only poll its cache memory system, resident on the hardware device itself, until such time as a data invalidation occurs. Once invalid, the hardware device reads the updated data from the main memory and begins using the data or executing the commands.

[0013] While the hardware is described as having a cache memory system, this cache memory system need not necessarily be the fast cache memory normally used for microprocessors, but instead may simply be random access memory of some kind, or if the FIFO buffer is small enough, a series of registers. Thus, the preferred embodiments of the present invention overcome the problems of the potentially unbounded latency software doorbell signal by allowing the hardware devices to participate, on a limited basis, in the cache coherency protocol using the invalidation signal of the cache coherency system as the notification.

[0014] The disclosed device comprises a combination of features and advantages which enable it to overcome the deficiencies of the prior art devices. The various characteristics described above, as well as other features, will be readily apparent to those skilled in the art upon reading the following detailed description, and by referring to the accompanying drawing.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawing in which:

[0016] Figure 1 shows an exemplary implementation of a system where a hardware device participates in the cache coherency domain.

NOTATION AND NOMENCLATURE

[0017] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may

refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function.

[0018] In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0019] Figure 1 shows a computer system 100 constructed in accordance with the preferred embodiment. In particular, computer system 100 preferably comprises a plurality of microprocessors or central processing units 102A-D. Although only four such microprocessors 102 are shown in Figure 1, the computer system 100 may comprise any number of microprocessors, and still be within the contemplation of this invention. Although having multiple microprocessors 102 is the preferred implementation, it would be possible to implement the techniques described herein in a single microprocessor 102, with cache coherency maintained between the single microprocessor's cache, the main memory, and the cache of the hardware device. Each of the microprocessors 102 preferably couples to a host node logic 104 by way of respective local buses 106 and 108. Inasmuch as the microprocessors 102 are preferably the same type, likewise the microprocessor buses 106 and 108 preferably implement the same bus protocol. The preferred microprocessors 102 are any of the 64 bit processors made by Intel®; however, any microprocessor capable of parallel operation in a computer system could be equivalently used.

[0020] The main memory array 110 preferably couples to the microprocessors 102 and the rest of the computer system 100 through the host node 104. The host node 104 preferably has a memory control unit (not shown) that controls transactions to the main memory array 110 by asserting the necessary control signals during memory accesses. The main memory array 110 generally comprises a conventional memory device or array of memory devices in which program instructions and data may be stored. The main memory array 110 may comprise any suitable type of memory such as dynamic random access memory (DRAM) or any of the various types of DRAM devices such as synchronous (SDRAM), extended data output DRAM (EDO DRAM) or RAMBUS™ DRAM (RDRAM).

[0021] The preferred embodiments of the computer system 100 also comprises an input/output (I/O) bridge 112 coupled to the host node 104 by way of a primary expansion bus 114. Any suitable primary expansion bus 114 may be used, and thus various buses such as the Hublink bus proposed by Intel Corporation or a peripheral component interconnect (PCI) bus may be used as the primary expansion bus 114..

[0022] The input/output (I/O) bridge 112 preferably bridges the primary expansion bus 114 to one or more secondary expansion buses. In the preferred embodiment, the secondary expansion bus 116 is a PCI bus, or its improvement, the PCI-X bus. However, computer system 100 is not limited to any particular type or number of secondary expansion buses, and thus various buses may be used in the secondary capacity, including an industry standard architecture bus (ISA), a sub-ISA bus, a universal serial bus (USB), and IDE bus, an IEEE 1394 standard ("Firewire") bus, or any of a variety of other buses that are available or may become available in the future.

[0023] Although Figure 1 only shows one host node 104 coupling the I/O devices to the central processing units 102, it must be understood that computer system 100 could also include a plurality

of host nodes 104, each host node coupling a plurality of central processing units and each having a memory array 110 coupled thereto. Such a multiple host node computer system would further require a switch logic coupling the host nodes to the I/O bridge for information routing purposes.

[0024] Figure 1 also shows a hardware device 118 coupled to the secondary expansion bus 116. This hardware device 118 could be any of a number of possible devices, including network interface cards, video cards, audio devices, data storage devices, system area network interfaces (*e.g.* Infini Band), storage area network interfaces (*e.g.* Fibre Channel) and any device capable of bus-mastering to and from the main memory array 110. Many of these hardware devices 118 need to communicate with programs executed on one or more of the microprocessors 102. While it is certainly possible to have the software stream communicate directly with the hardware device 118 by writing directly to the hardware device, the commands and/or data may be too lengthy for the hardware device 118 to accept at any one time, or there may be latency problems in the communication as discussed in the Background section. To combat this problem, the command lists and/or data are preferably placed in a first-in/first-out (FIFO) buffer 120, which may also be referred to as an exchange buffer, located in the main memory 110. The software stream merely places the commands and/or data in the FIFO buffer 120, and subsequently the hardware device arbitrates for a bus mastership of the secondary expansion bus 116 and reads the commands and/or data from the FIFO buffer 120. A discussion of how, in the preferred embodiment, the hardware device 118 is notified of the presence of commands and/or data in the FIFO buffer 120 requires a brief digression into memory coherency protocols.

[0025] In a system having multiple microprocessors 102, and preferably with each microprocessor having at least some cache memory (either internal, L1, cache memory and possibly external, L2, cache memory), there is a need to insure cache coherency across all the

caches for all the microprocessors with respect to the main memory, and vice versa. Consider for purposes of example a software stream executed on CPU 102A of Figure 1. Further consider that the software stream updates a variable from main memory, a copy of which is present in the cache of the CPU 102A. By writing a new value to the cache memory location, the cache memory version becomes the only valid version within the system. Cache coherency protocols are responsible for propagating the new value to all the appropriate locations, or at least notifying other CPUs 102 that their copy (if they have one) of the parameter is invalid. While there may be several possible cache coherency protocols that could ensure this coherency, in the preferred embodiments the cache coherency protocol is a write-back invalidate protocol. In a write-back invalidate cache coherency protocol, each agent wanting to modify memory must seek and obtain modify rights prior to the modification of the memory location. In being granted modify rights by the coherency system, other shared copies memory location are invalidated. Consider for purposes of explanation a piece of memory shared between two microprocessors. In this initial state, the status of the memory location is shared valid in each microprocessor. Further consider that a first processor seeks to modify the memory location, and thus requests permission from the device implementing cache coherency for this permission, in the preferred embodiments host node 104. The host node 104 grants permission to modify the memory location (gives the requesting microprocessor exclusive ownership), and simultaneously invalidates other copies, in the exemplary case the duplicate held in the second microprocessor. The microprocessor having exclusive ownership may change the value of the memory location (or rather the version stored in the microprocessor's cache) at will. It is not until the memory location is evicted from the requesting processor's cache, or some other device (the second microprocessor or other device capable of running memory transactions) requests the data at the memory location, that the updated

value is written back to main memory. A write-back invalidate cache coherency protocol is preferred because write through cache protocols, requiring each cache line modification to be written back to main memory, are not compatible with the 64 bit machines offered by Intel®.

[0026] It is standard in the industry to define a coherency domain to comprise all devices within the computer system that receive an invalidation notice for copies of data which they contain. It is clear that the microprocessors all operate within the same coherency domain, preferably maintained by the host node logic 104. Other devices, such as hardware device 118 coupled to the secondary expansion bus 116, have traditionally not been allowed to participate in the coherency domain of the microprocessors. However, the preferred embodiments of this invention are directed generally to allow hardware devices, such as the hardware device 118, to participate in the coherence domain with respect to the FIFO buffer 120.

[0027] In the preferred embodiments the hardware device 118 has an onboard cache memory 122 (hereinafter cache 122). This cache 122 preferably duplicates information stored in the FIFO buffer 120 of the main memory 110. In broad terms, the cache memory on the hardware device is treated like a cache memory in one of the microprocessors 102, and is kept coherent therewith. Thus, if a software stream executed in one of the microprocessors 102 updates or places commands and/or data in the FIFO buffer 120, the cache coherency protocol, preferably implemented in the host node logic 104, sends invalidation messages to the hardware device 118 indicating the invalidation of one or more cache lines, preferably 128 bytes of data, duplicated in the cache 122. Upon receiving the notification that a cache line has been invalidated, the hardware device 118 preferably arbitrates for bus mastership of the secondary expansion bus 116, and reads the new data from the main memory 110 FIFO buffer 120. Thus, notification that commands and/or data are available with the bounded latency of the invalidation commands of the cache coherency

protocol. Software need only be concerned with writing the commands and/or data into the FIFO buffer 120. The cache coherency protocol is responsible for sending invalidate commands to the hardware device 118, which the hardware device 118 uses as a notification that commands and/or data are available.

[0028] More particularly, the host node logic 104 preferably has a series of registers 124, 126 and 128. The registers 124, 126, 128 preferably identify the top of the FIFO buffer, the bottom of the FIFO buffer, and a destination respectively. The top 124 and the bottom 126 registers simply indicate the range in main memory of the location of the FIFO buffer 120. Inasmuch as the FIFO buffer 120 is preferably a linearly addressed set of memory locations within the main memory 110, the location may be completely and uniquely identified by having the top address and bottom address. Alternatively, the register 124 could contain a starting address, and register 126 could contain an offset indicating the length of the FIFO buffer. Operation of the registers 124, 126 and 128 of the preferred embodiment is best described with regard to an exemplary write of information to the FIFO buffer 120. In particular, consider a software stream executed on one of the microprocessors, for example microprocessor 102A. Further assume that the microprocessor 102A does not have a copy of the FIFO buffer 120 in its onboard cache (not shown). Thus, the microprocessor 102A copies portions of, or the entire, FIFO buffer from the main memory 110 to be placed in its cache. Further, because the software intends to update these locations, the microprocessor 102A requests of the cache coherency protocol exclusive ownership of those memory locations. Preferably, the host node 104 compares the addresses for which the microprocessor 102A requests exclusive ownership to the top and bottom registers 124 and 126 respectively. Because all or a part of the FIFO buffer 120 is preferably duplicated in the cache memory system in the hardware device 118, this comparison of the addresses to the registers 124

and 126 preferably reveals that the hardware device 118 likewise has copies of these location. The cache coherency protocol preferably simultaneously grants exclusive ownership of the memory locations to the requesting microprocessor 102A, and invalidates the copies held in the hardware device 118 by sending in an invalidate command to the location identified by the destination register 128.

[0029] In the preferred embodiments, the destination register 128 identifies the I/O bridge 112. Upon receiving the invalidate command from the host node 104, the I/O bridge 112 compares the addresses of the invalidate command to its top and bottom registers 132 and 134 respectively. These registers are preferably beginning address and ending address respectively, but may also be a beginning address and an offset. Preferably, these registers indicate that the device indicated in register 136 is the hardware device that contains the duplicate copies, and the invalidation is relayed across the secondary expansion bus 116 to the indicated hardware device, in this case hardware device 118. This invalidation command preferably changes the state of the valid bit 138 for each cache in the cache memory 122 of the hardware device 118 for which the invalidation command pertains. At this point, the microprocessor 102A, having exclusive ownership, is free to repeatedly change or add to those cached main memory locations. Whether by eviction of the relevant data by the microprocessor 102A, or a request to read that data from the hardware device 118, the microprocessor preferably writes the updated data to the main memory locations holding the FIFO buffer 120.

[0030] The hardware device 118 of the preferred embodiments, rather than continuously polling the actual main memory FIFO buffer 120, polls its cache looking for valid commands and/or data. Because the hardware device participates in the cache coherency domain, the hardware device 118 need only look at the version within its cache memory 122. As soon as that hardware device 118

polls the data in its cache 122 and sees that it is invalid (as caused by the invalidate command propagating down from the host node logic 104), the hardware device preferably arbitrates for ownership of the secondary expansion bus 116, and reads the new data from the main memory 110 FIFO buffer 120. After the data resides within the hardware device's cache 122, the device performs the command specified or operate on the data as required. Thus, the invalidate command sent by the host node logic 104 acts as the notification to the hardware device 118 that commands and/or data are available in the FIFO buffer 120.

[0031] Communication of data from the hardware device 118 to the software stream preferably takes place through a buffer in main memory. In particular, the hardware device places data, through its bus-mastering and direct memory access capabilities, in a second FIFO buffer (not specifically shown) in the main memory 110. By polling the next memory location, the software determines when the data transfer has taken place. Thus, the hardware device does not cache this second FIFO buffer in the preferred embodiments.

[0032] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.